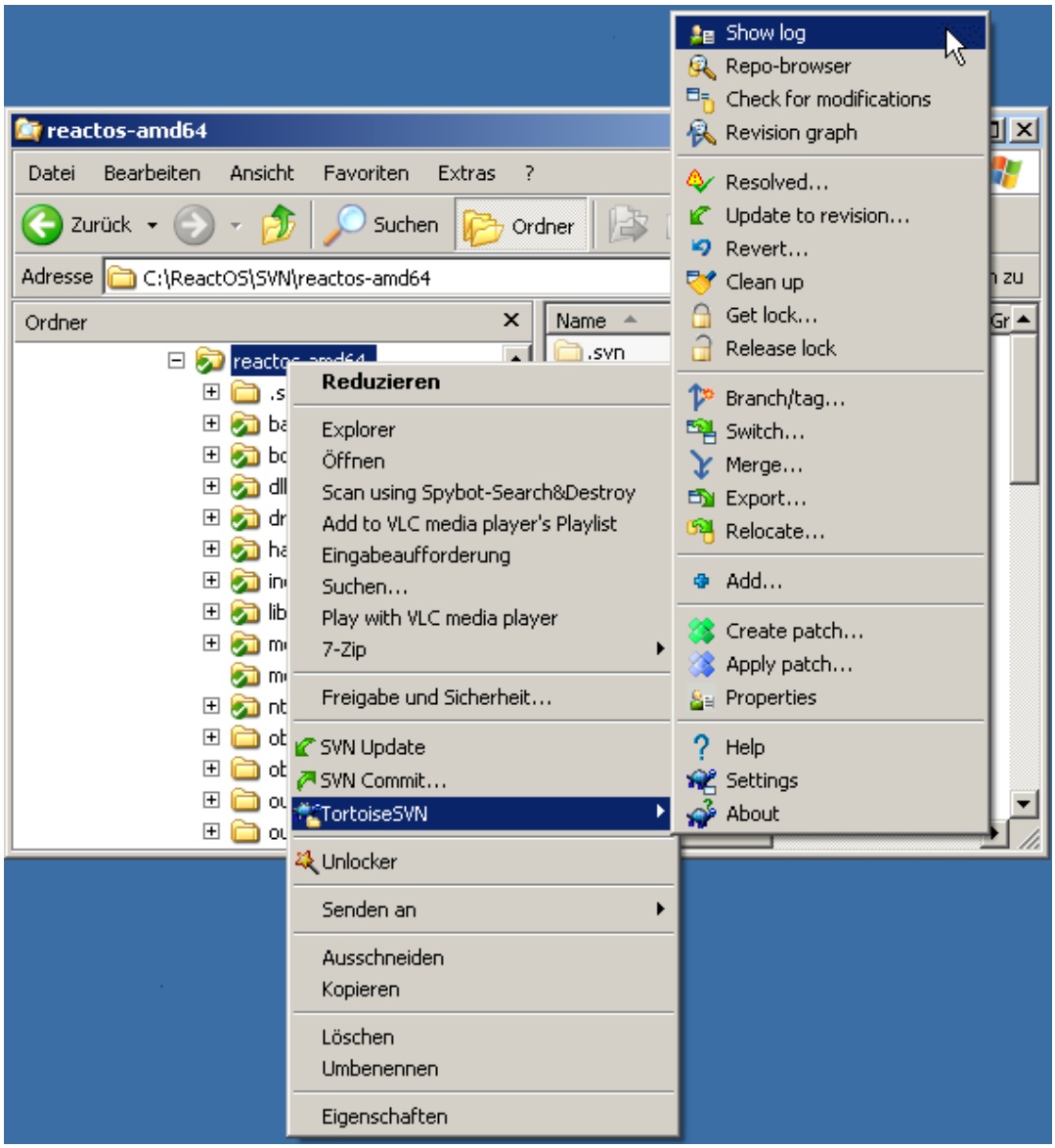
Version Control System: Git/SVN/(TFS)

	Centralized Version Control System(Subversion-SVN)	Distributed Version Control System(GIT)
Difference	Only stores the snapshot of the current version in local	Stores the whole history of versions in local
	The repository is located at one place and provides access to many clients.	every user has a local copy of the repository in addition to the central repo on the server side
Pros	CVCS is easy to administrate and has more control over users and access as it is server from one place.	DVCS provides the benefits to **work offline**. Everything except push and pull the code can be done without an internet connection. **Save time** as doing local comparations
	CVCS allows you to checkout only few files of code if you just need to work on few modules. (Save space if project has a long history)	DVCS provides an advantage wherein if the main server's repository crashes, you still have a local repository in every developer's local space from which you can create the main repository.
Cons	Lose all data if the central repository is down.	
	Developers cannot work if Central Repository stop working.	

Git log

A(initialize project) -> B(change title name from “1” to “2”) -> C(change title name from “2” to “3”)

Tortoise SVN



1.remote and local

Git Remote

Git pull(update local from remote) Git Push(update remote from local)

Git Local ===> making modification

REMOTE —----------

LOCAL —----------

2. Git Status (buffer stage):
Local files:
Original ===> saved files ===> staged (changed files is recorded in GIT)/unstaged (git add filename) ===> commit change(final decision), git commit

A/B/C/D/E/F ===> staged A/B ===> the git is tracking for file A/B ===> git commit

.git

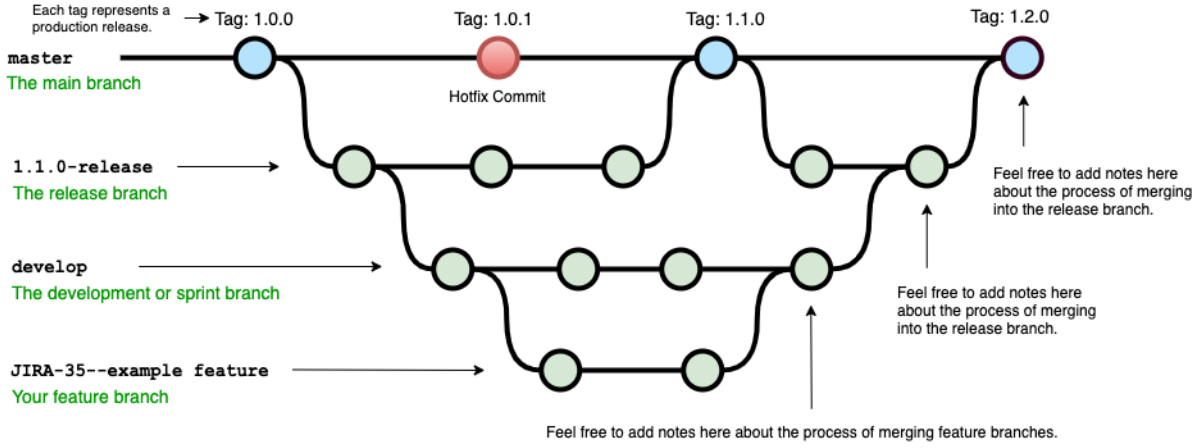
Git Branch Diagrams

Git clone path ===> Git checkout branchName===> git checkout -b newBranchName ===> git add filename ===> git commit -m “description”===> git push ===> pull request (pr)/ merge request(code review) ===>git merge (you need to stay in the branch you want to merge to) branchName(the branch you want it to be merged)

Example Git Branching Diagrams

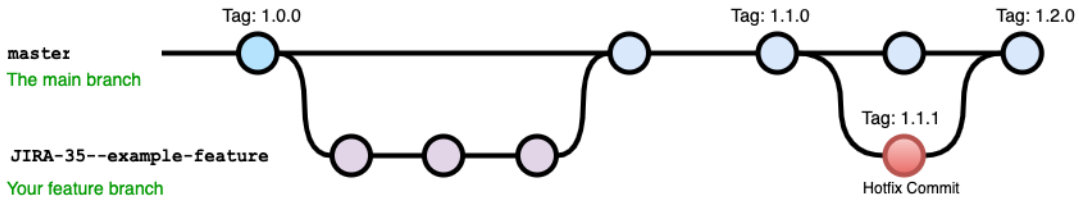
Example diagram for a workflow similar to "Git-flow" :

See: https://nvie.com/posts/a-successful-git-branching-model/



Example diagram for a workflow with a simpler branching model:

See: https://gist.github.com/jbenet/ee6c9ac48068889b0912 or https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow



Git commands:

Git clone projectPath : clone the project
Git repo name : project1
Run git clone under folder : antraProject/ project1/ project1.git
By default: you are at master branch

Git checkout ReleaseBranchName Naming Standard: Release_Branch/Team_name
Switched to the release branch
Git pull(update your local branch from remote)

Git branch featureBranchName : stay in the current branch
(git checkout -b featureBranchName) : switch to the new branch

Naming Standard:
In real world:
Release/{{Release_version}} : Release/1.0.0
Feature/{{Ticket_Number}}/{{summary}} : Feature/FEB23-1/test_ticket
Bugfix/{{Ticket_Number}}/{{Summary}}
//

Release Branch: Release_Branch/Team_Name

Feature Branch: Feature/Team_Name/Your_Name/Ticket_number/Summary

bugFix/Angular-123_fix_bug_BCD

Staged files

Git add fileName
will staged files which will be recorded in the next commit

Git commit

Commit : the changes will be recorded into the commit

Developing : when you make all the changes in the code without commit, it will not be recorded in git

---

Git push: update your remote branch from your local branch
(publish the branch to remote: git push --set-upstream origin featureBranchName)

Pull request/ merge request — code review

Git merge branchName (you should stay in the branch you want to merged to)
In release branch: git merge featureBranch

Create pull request/ merge request — PR/MR

Code review:

(pull the code and check if the project is able to be built without compiling error and all the functionalities work properly)

1. Readability of the code: Naming of folders/ files/variables/function
2. Formatting issues: space indent, new line
3. Try not to hard code (create a contant.ts ,global variable file)
   Constant.file

   ```
   Export class constant {
       Const themeColor = lightblue;
       Const admin_role = 1
   }
   if(role === constant.admin_role)
   ```

4. Try to create reusable component as much as you can
5. Do not write inline style
6. Sort the css selectors/ js functions/ variables

Git technics:

Git branches

Normal:

Initialize project : A -> B =>“description”,(fileA, line 1,column 50, from empty to “hello world”; fileB, line1, from empty to “test”) -> C => “description,(fileA, line1, from “hello world” to “hello Antra”)”

Master: A->B->C

Release Branch : A->B->C->D->E

Feature Branch: A->B->C->E->F->G commit behind/ commit ahead
//
Remote Feature Branch: A->B->C->D->E->X->Y->Z

Local Feature Branch: A->B->C->D->E->X->Y->Z->F
//

Duo merges with conflict:

Release Branch : A->B->C->D->E(FileA line1 from “name” to “lastname”)->F(FileA line 1 from “lastname” to “firstname”)

Feature Branch 1: A->B->C->D**->E(FileA line1 from “name” to “lastname”)**

Feature Branch 2: A->B->C->D->F(FileA line 1 from “name” to “firstname”)

//

Duo merges without conflict:

Feature Branch 2: A->B->C->D->G->H
Release Branch : A->B->C->D(file A line 1 : name)->G(change file A line 1: name => firstName)->H

Feature Branch 1: A->B->C->D->E(change file A line 2:age => school age)->F

A->B->C->D->G(change file A line 1: name => firstName: based on commit D)->H->E(change file A line 2:age => school age: based on commit D)->F

//
Git pull === updating the code from remote to local within the same branch

Release Branch : A->B->C->D ->E->F

Feature Branch: A->B->C->D ->G->H

//
Release Branch : A->B->C->D->E->F

Feature Branch: A->B->C->D === delete

Create new one based on the latest version of Release Branch

Inside feature branch: git pull remote_release_branch

//

In order to avoid the code conflicts, try not to assign the same file tasks to different developer

//

How to solve the conflicts

Small conflicts:

1.Manually fixed the code by modifying the conflict version

2.Branch A merge firstly, branch B merge secondly(causing conflict)

If branch B does not have a lot of changes => close pull request for branch B

Update release branch, then create new branch C based on the latest version of release branch. Then apply B's change to C

Huge conflicts:

1. If you have more than 1 branches that solving the tickets, then just choose one of them
2. If you are working with the same file: discuss with your team. Rewrite the code or reset/revert the commit version

//

Git reset vs git revert

A->B-C->D("name" to "lastname")->E("name" to "firstName")

Feature Branch 1: A->B->C->D->E(false commit: changing from "name" to "lastname")->F(we are applying changes based on False commit changes) — have to use git reset

git reset commit_D ====>Feature Branch 1: A->B->C->D

Feature Branch 1: A->B->C->D->E(false commit)->F :
Git revert commit_E ====> Feature Branch 1: A->B->C->D->E(false commit)->F->revert_E

//

Git merge and Git rebase

Release Branch : A->B->C->D

Feature Branch: A->B->C->D->E->F->G

Git merge featureBranch
Git merge : feature to release
Release Branch will be : A->B->C->D->mergeNode_Z

Git rebase: feature to release
Release Branch: A->B->C->D->E->F->G

//

Git pull vs Git fetch

Git pull = Git fetch / Git merge

Git pull : update code from remote to local ===> all the codes in all files are updated
Git fetch : update the change log from remote to local

Git checkout commitNumber